COVER SHEET

A FRAMEWORK FOR EXECUTING COMPUTATIONAL INTELLIGENCE OVER DISTRIBUTED EMBEDDED NODES

Robert A. Ellen and Duncan A. Campbell
School of Engineering Systems
Queensland University of Technology, Australia
email: r.ellen@student.qut.edu.au, da.campbell@qut.edu.au

ABSTRACT
Manufacturing and mining automation, robotics, swarms and smart device networks are often implemented upon distributed embedded systems. These systems are typically statically distributed, coarsely reconfigurable or deployed on homogeneous networks. A conceptual stack can be formed using modelling languages, system performance analysis and optimisation and reconfigurable platform-neutral components to overcome these problems. A framework for computational intelligence-based applications to be built upon this stack has been proposed. Model-Driven Architecuture has been shown to be a promising standard for the modelling, design and development of embedded applications. The Theory of Constraints is proposed as a potential technique for performance analysis and optimisation of distributed systems. Mobile Agents and code mobility can be used in the component architecture to allow for adaptation and reconfiguration for optimisation.

KEY WORDS
Distributed and embedded systems, system design automation.

1 Introduction

Distributed embedded systems can be found in a range of problem and solution domains such as manufacturing automation and control, robotics, mining automation, swarm intelligence and smart device networks. The modelling, design, implementation and optimisation of these systems are important areas of research for engineers and computer scientists alike.

Theoretical frameworks based on technologies such as Petri Nets [1,2], the Unified Modeling Language (UML) [3,4] and other modelling techniques [5,6] have been used to model, analyse, specify and design software applications for distributed and/or embedded systems. Techniques based upon statistical and model control have been used to analyse and optimise the performance of distributed, embedded software [7,8]. Mobile agents and code mobility have allowed researchers and practitioners to develop software that is adaptable and reconfigurable [9,10].

These research topics are often considered in isolation and do not necessarily take advantage of the contributions made in other areas. Consequently, distributed embedded systems are often restricted in some way. The distribution of tasks over the network may be statically defined, reconfigurability may be manual or only available at a coarse level and the platforms in use may be homogenous.

When the modelling, design and dynamic reconfigurabilty of distributed embedded systems are considered together, these research efforts form a *conceptual stack* that can overcome the problems of isolated application. Currently, research is being undertaken into computationally intelligent (CI) applications implemented upon such a stack. The goal is to develop a holistic framework for the modelling, design, execution and optimisation of CI applications distributed across heterogeneous networks of nodes that have a broad range of processing power, memory size and communication bandwidth. A range of CI applications, too complex to run wholy on any single node, will be able to be distributed effectively across the network, which may include PCs, programmable logic controllers (PLCs), embedded microcontrollers, digital signal processors and reconfigurable hardware.

Potential manufacturing control applications range from small isolated implementations of CI up to holonic or flexible manufacturing systems and plant-wide intelligent systems such as those described in [11]. Applicable mining automation problems include underground longwall mining equipment automation [12] and underground conveyor and bolting machine automation [13].

This paper proposes a framework for CI applications on distributed embedded systems, reviewing the theory regarding it's proposed structure. The next section covers the modelling and design techniques that are being considered. Section 3 outlines the autonomic reconfigurability for optimisation strategy. Section 4 discusses the proposed implementation of the framework in terms of component architecture, communications and mobility. The concise framework proposal is discussed in section 5.

2 Modelling and Design

The framework will include a set of modelling languages and a design environment to support the development of applications. Structural and behavioural modelling will be used to develop and refine the atomic CI components and the intelligent supervisory components (see section 4). A functional modelling and design environment will be used

to compose the CI applications from component libraries. Finally, to integrate the modelling of the components and applications together, as well as develop applications with conflicting constraints and concerns, an overarching modelling and development philosophy must be taken.

2.1 Modelling and Design Languages

UML has become the defacto standard for modelling the structure and behaviour of object-oriented software systems [14]. The key tools in UML are class diagrams, interaction diagrams (depicting the flow of control between objects) and state diagrams. Embedded systems modellers generally extend basic UML to incorporate real-time capabilities, as seen in such works as [3, 4, 15].

The theory of Petri Nets, complimentary to UML, is a tool for behavioural modelling of multi-state automata. Examples of petri net use in modelling and analysis of embedded systems include Stochastic Petri-Nets [1], Predicate-Transition Nets [2] and Timed Petri-Nets [16].

Functional modelling and design languages often manifest themselves as block diagram tools, the blocks encpasulating some function and the connections representing data and control flow. A popular funtional modelling tool seen in research is MATLAB/Simulink and Ptolemy II [5]. A functional standard seen in automation is the IEC 61499 function block PLC language [9].

2.2 Model-Driven Architecture

A range of languages and tools are to be supported for development of CI applications. In addition, different applications may only require some of the methods described. To provide this range and variability, a nexus that cohesively brings together the modelling and design tools is required. An identified candiate is the emergent concept of Model-Driven Architecture (MDA). Developed by the Object Management Group (OMG), Model-Driven Architecture is an overarching methodology with the following goals [17]:

- raise the level of abstraction at which software development occurs (from compiling source-code to compiling models)
- increase software re-use
- provide software interoperability at design-time
- allow software design models to become assets

These goals align directly with the goals of a framework for CI applications.

MDA is new and by no means generally accepted in the software development community. The agile software development community especially has concerns , comparing MDA to the past hype surrounding Components (see section 4) and Fourth Generation Languages (4GLs) [18]. Another point of contention is the heavy use of UML by MDA as opposed to domain-specific modelling languages (DSML) [19]. Despite this there are several examples of MDA (or very similar techniques) applied in embedded systems [6, 20, 21].

The key realisation that can make MDA successful in embedded systems and CI is that both problem and solution domains are specific. The problem at hand is the development of CI applications, as opposed to generic applications. To build these applications the subset of MDA most directly useful comprises metamodelling (creation of domain-specific modelling languages), platform-independant to platform-specfic model mappings, model marking (comparable to aspects) and executable models [17]. With these tools, a heterogenous CI application can be specified from a set of structural, behavioural and functional models as well as associated aspects and marks covering platform-specific issues and non-functional constraints. A full implementation of MDA, attempting to be all things to all people, is not required to reap the benfits from its adoption.

MDA is also a candidate for a design-time *Seperation of Concerns* framework to deal with the multiple views from which the specification and design of a system can be viewed. As well as the disparate concerns of different modelling methods, non-functional requirements such as performance, power consumption or memory footprint are important considerations that permeate throughout embedded applications. The application of seperation of concerns at design-time has been proposed as a method for reducing the complexity of embedded systems design. de Niz et al. [6] and Stankovic et al. [21] present embedded systems frameworks and the tools that seperate concerns such as timing, concurrency, fault-tolerance and memory footprint and schedulability checking.

3 Reconfiguration for Optimisation

Fine-grained, autonomic reconfiguration capabilties allow for optimisation (in dimensions such as performance, power consumption, communications and footprint) of the application to suit the changing network characteristics. Reconfigurability facilitates a degree of fault tolerance by allowing the functionality located upon faulty or non-responsive to be moved and re-routed. A reconfiguration request may involve the ‘recompilation’ of that part of the application so that it can be run on a node of different characteristics to those of the original execution location. This service would be provided by an on-line MDA process. The decision to perform a reconfiguration of an application for optimisation can be based upon evidence from an on-line analytical or emergent (i.e. complex systems) technique.

3.1 Analytical Optimisation

Static analytical approaches to improve performance, such as rate analysis, discrete event analysis and queing theory can be used to predict performance prior to deployment. The framework, at the modelling and design level, will accommodate such analysis in its specification of the initial deployment of the application components to meet performance constraints.

Run-time analytical methods used to optimise the performance of distributed, embedded systems include statistical control [7] and model predictive control [8] based on design-time analysis and CPU utilisation feedback. These methods can be applied to highly pipelined applications, examples of which include fuzzy inference systems or neural nets.

A technique not known to be applied to performance optimisation of pipelined computational systems is the *Theory of Constraints*. This optimisation methodology (not to be mistaken for constraint satisfaction problems), shares a mathematical basis with queue theory and originates from pipelined manufacturing or supply-chain systems [22]. The core idea in the Theory of Constraints (TOC) is the intuitive realisation that the throughput of a chain of pipelined processes is limited to the throughput of the slowest process. This step is identified as the *system constraint* or *capacity constrained resource*, and any optimisation effort should be directed towards increasing its performance so that it is no longer constraining the pipeline. This optimisation procedure is followed repeatedly until the inputs and/or outputs of the system are the constraints.

3.2 Emergent Optimisation

Optimisation based on emergent behavioural techniques may be beneficial due to lower computational overheads or elimination of human design bias. A neuro-fuzzy model predictive controller based on processor utilisation (in a similar configuration to Kang et al. [7]) or a rule-based complex adaptive system (as in ant colony-type systems) are possible emergent techniques.

4 Component Implementation

The core artifacts of the framework will be abstract components that are used to develop the CI applications. Examples of CI components range from Sigmoid functions or fuzzy membership functions up to entire multilayer perceptrons or fuzzy interence systems, as well as encapsulations of system inputs and outputs and other services. A component architecture must be developed to support realisation such components, in both software and hardware. The communications between components must be sepcified, with the heterogenous nature of potential target environments. To support reconfigurability, components within a CI application must have the ability to move about the network of nodes at run-time.

4.1 Component Architecture

Within a CI applications framework, as in software engineering in general, a component is some unit of deployment that has no persistant state which is used to compose an application [23]. A component may be indivisible or may be composed of other components. The overall application would comprise many component 'instances' (which may have persistent state) connected together using well known standard interfaces implemented by the components [24]. These connections must have the potential to be intra- or inter- node.

There are numerous examples of component architectures and associated middleware for distributed, embedded systems, such as Port Based Objects [24], Actors [5] and BASE Microbroker [25]. Key descriminating factors of such component architectures include the inter-component communication methods (such as shared memory or message passing), concurrency (whether or not a component has its own thread of control) and location transparency (one component is unaware of the physical location of another component).

Figure 1 shows the proposed component architecture and middleware. Each node will have one or more containers, comprising the set of components that run within them and a message-router for inter-component communications. The components include ports for I/O and control.

4.2 Communications

The likely physical communications networks that the framework will encounter include statically defined or ad hoc. Static networks may include the ubiquitos TCP over ethernet, commodity buses (for example USB and IEEE 1394) or a number of industrial busses, such as ControlNet, Modbus or CAN. *Spontaneous networks*, such as swarms or other mobile nodes will likely comprise wireless links such as infrared, Bluetooth or IEEE 802.11. Network services include SUN's Jini, CORBA, IIOP, which are most commonly implemented on top of TCP/IP, will provide the integration between the physical channels and component middleware and CI applications and support transient network connections. The factor that will play the most significant role in the determination of the lowest-common-denominator in terms of communications will be the ability to protect the network of nodes from security breaches. A spontaneous network configuration (without physical protection against outside attackers) would require strong cryptographic abilities as well as safe operating modes in the event of Denial of Service attacks or loss of network connectivity.

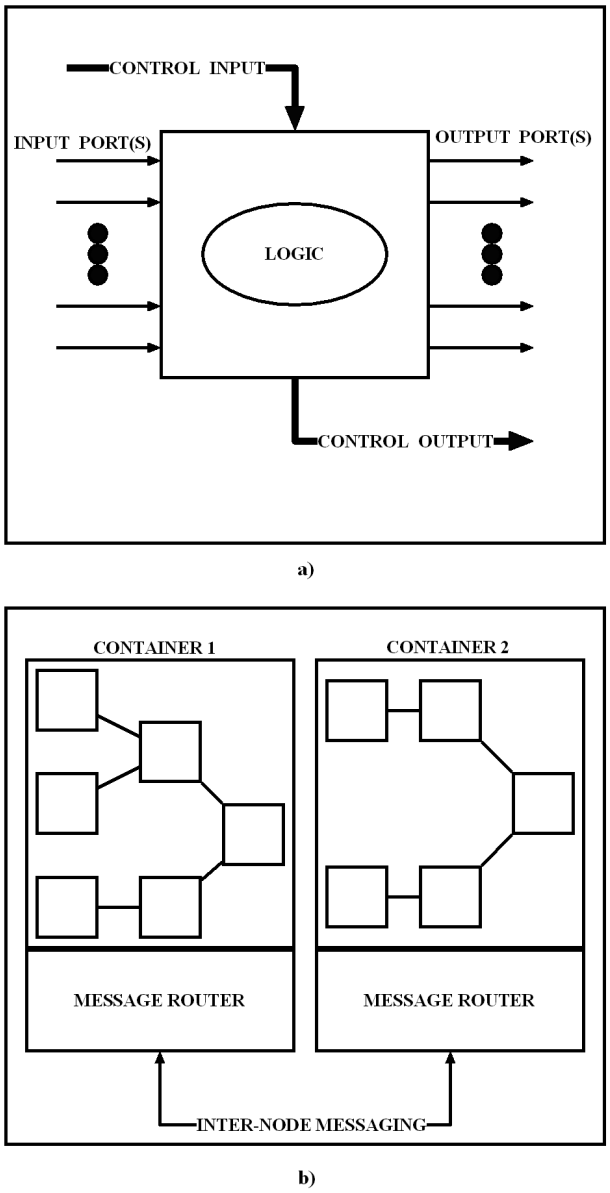

Figure 1. Proposed Component Architecture showing a) an individual component and b) container architecture

4.3 Component Mobility

The two areas of research that are applicable in the implementation of reconfigurability are those of *Mobile Agents* and *Code Mobility*. Mobile Agents are those agents (generally software or logical as opposed to physical) that can change the location of their execution. Mobile agent implementations have been reviewed in such works as [26] and [27].

A drawback to the wholesale and exclusive application of Mobile Agents to a CI framework is that use of agents implies a coarsely-grained reconfigurability. This arises because agents tend to have a greater perception of their environment and require storage and processing power for facilities such as knowledge-bases and planning. Agents are applicable as intelligent supervisors for reconfigurability and optimisation [9] or as facilitators of spontaneous networking [28] however individual components implemented as mobile agents add considerable overheads.

A better option for component movement is Code Mobility, which refers to fragments of a complete program that can change their location of execution. This implies that code mobility is more fine-grained than mobile agents [29].

For a framework that does not specify whether a component is a software or hardware construct, code mobility must be generalised into *component mobility* [30], allowing components to be moved between software and hardware based nodes dynamically. This reinforces the concept of seperation between a component's interface and its (potentially) multiple implementations.

5 Discussion

The desired outcome of the development of the framework is the ability to develop distributed embedded CI applications end-to-end. Considering seperate theory that is usually applied in isolation as a whole will assist in this aim. Given the presented overview of this conceptual stack, the proposed framework is structured as follows:

- the spirit of MDA will be used in the modelling, design and compilation of CI applications
- TOC with modification will be implemented as the performance evaluating and optimising technique
- mobile agents will be used to monitor performance and actuate reconfiguration
- mobile components and location-transparent communications method will be used to implement the computational intelligence components

Initially, there will be some limitations upon the framework's capabilities. Components will be developed in software only. Future research will investigate the integration of existing reconfigurable hardware-based frameworks such as the work of Williams and Bergmann [31]. Additionally, real-time constraints will not be considered to begin with. It must be noted however, that most research into distributed, embedded systems (including that presented here) concentrates upon *distributed real-time embedded systems* (DREs). As such, real-time constraints can be introduced in the future as a seperate concern using the wealth of research available.

To investigate TOC as an optimisation method, buffers will be incorporated into the component and communication impementation. The signs that reconfiguration is necessary to increase the performance of the application are buffer overflow at components before the constraint and buffer starvation at components after the constraint.

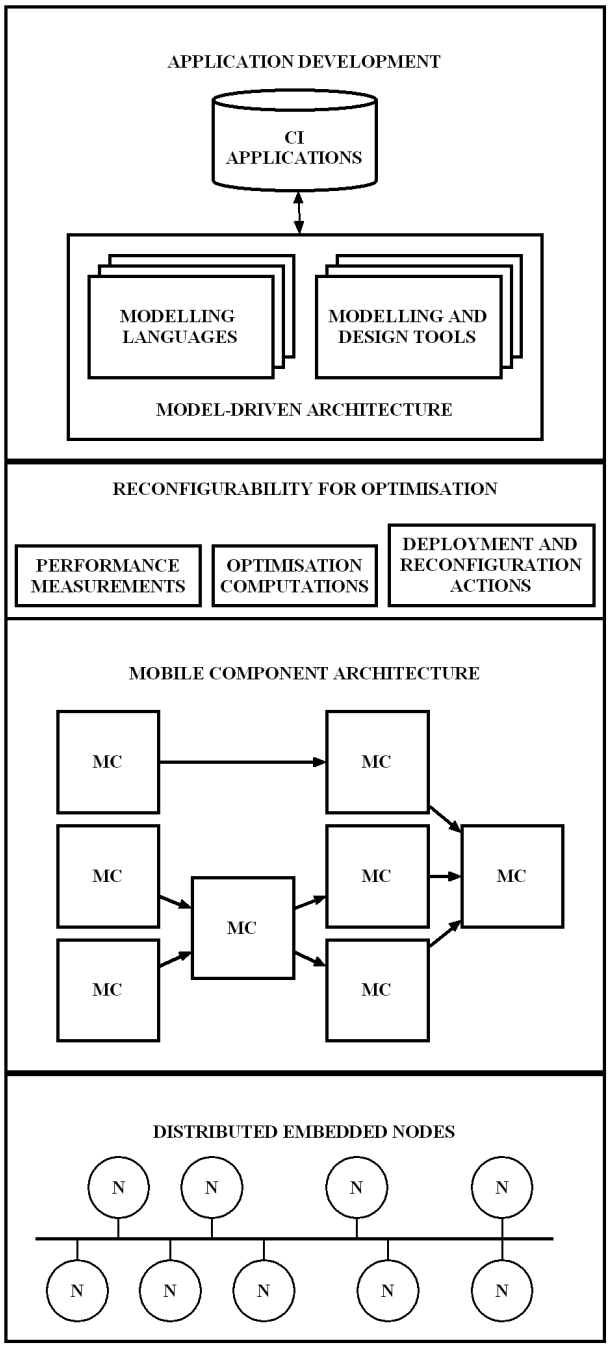

Figure 2. Proposed Framework

In terms of TOC, and the mobile code architecture of the framework, the options to increase performance include replacement with as faster implementation, upscaling to a more powerful (or less utilised) node, or cloning (parallelizing) the component. The cost of such actions is the communications and synchronisation overheads incured.

While TOC, as applied to production lines, is mainly used for performance optimisation, it can be inverted to allow for optimisation of footprint or power consumption. A process that is starving could be downscaled to a less powerful node, consolidated with other processes or merged (if it had previously been parallelized).

A key question to be asked is whether this method is robust enough to optimise a range of CI application configurations. TOC optimisation may only be suitable for more pipelined applications. To answer this, a range of strategies for optimisation must ultimately be investigated.

6 Conclusion

A range of automation applications involving computationally intelligent techniques are implemented with distributed, embedded systems. These include manufacturing, robotics and mining. Distributed, embedded systems may also form part of intelligent swarms or smart device networks.

Research efforts involving such systems include modelling and design, autonomic reconfiguration for optimisation and the associated underlying mobility of the systems' components. Limitations of considering these topics in isolation include static distribution, course-grained and manual reconfiguability and homogenous platform dependance. The realisation has been made that these areas of research are related and can form a conceptual stack that works towards overcomming these limitations.

A framework, based upon this conceptual stack, has been proposed for developing and executing computational intelligence applications deployed upon distributed embedded systems. Model-Driven Architecture has been suggested as the foundation of the modelling and design, the Theory of Constraints as a strategy for optimisation and mobile agents and components as the framework's executable artifacts.

Acknowledgments

The authors wish to thank Mick Lees from Carlton & United Beverages, Australia for assistance in the preperation of this paper.

References

[1] B. Ravindran, L. Welch, and C. Kelling, "Building distributed scalable dependable real-time systems," in *Engineering of Computer-Based Systems, 1997. Proceedings., International Conference and Workshop on*, pp. 452–459, 1997.

[2] C. Ebert, "Experiences with colored predicate-transition nets for specifying and prototyping embedded systems," *Systems, Man and Cybernetics, Part B, IEEE Transactions on*, vol. 28, no. 5, pp. 641–652, 1998.

[3] G. Martin, L. Lavagno, and J. Louis-Guerin, "Embedded uml: a merger of real-time uml and co-design," in *Proceedings of the ninth international symposium on Hardware/software codesign*, (Copenhagen, Denmark), pp. 23–28, ACM Press, 2001.

[4] S. Lu, W. Halang, and R. Gumzej, "Towards a comprehensive environment for the engineering of embedded control systems based on uml," in *Industrial Technology, 2003 IEEE International Conference on*, vol. 2, pp. 693–698 Vol.2, 2003.

[5] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity - the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[6] D. de Niz and R. Rajkumar, "Time weaver: a software-through-models framework for embedded real-time systems," in *Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, (San Diego, California, USA), pp. 133–143, ACM Press, 2003.

[7] D.-I. Kang, R. Gerber, and M. Saksena, "Parametric design synthesis of distributed embedded systems," *Computers, IEEE Transactions on*, vol. 49, no. 11, pp. 1155–1169, 2000.

[8] C. Lu, X. Wang, and X. Koutsoukos, "End-to-end utilization control in distributed real-time systems," in *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pp. 456–466, 2004.

[9] R. Brennan, M. Fletcher, and D. Norrie, "An agent-based approach to reconfiguration of real-time distributed control systems," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 4, pp. 444–451, 2002.

[10] R. Brooks and T. Keiser, "Mobile code daemons for networks of embedded systems," *Internet Computing, IEEE*, vol. 8, no. 4, pp. 72–79, 2004.

[11] D. Campbell and M. Lees, "Soft computing, real-time measurement and information processing in a modern brewery," in *Soft Computing in Measurement and Information Acquisition* (L. Reznik and V. Kreinovich, eds.), vol. 127, Springer Verlag, 2003.

[12] ACARP Landmark Longwall Automation Project Homepage: available at http://www.longwallautomation.org, Accessed 26/2/2005.

[13] J. Ralston, D. Hainsworth, R. McPhee, D. Reid, and C. Hargrave, "Application of signal processing technology for automatic underground coal mining machinery," in *International Conference on Acoustics, Speech and Signal Processing and Applications*, (Hong Kong), 2003.

[14] M. Fowler, *UML Distilled*. Object Technology Series, Boston, USA: Addison-Wesley, 3rd ed., 2004.

[15] P. Green and M. Edwards, "The modelling of embedded systems using hasoc," in *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pp. 752–759, 2002.

[16] Z. Gu and K. Shin, "An integrated approach to modeling and analysis of embedded real-time systems based on timed petri nets," in *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pp. 350–359, 2003.

[17] S. Mellor, K. Scott, A. Uhl, and D. Weise, *MDA Distilled*. Object Technology Series, Boston, USA: Addison-Wesley, 2004.

[18] S. Ambler, "Agile model driven development is good enough," *Software, IEEE*, vol. 20, no. 5, pp. 71–73, 2003.

[19] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 145–164, 2003.

[20] R. Obermaisser and P. Peti, "A framework for rapid application development of distributed embedded real-time systems," in *EUROCON 2003. Computer as a Tool. The IEEE Region 8*, vol. 1, pp. 80–84 vol.1, 2003.

[21] J. A. Stankovic, P. Nagaraddi, Z. Yu, Z. He, and B. Ellis, "Exploiting prescriptive aspects: a design time capability," in *Proceedings of the fourth ACM international conference on Embedded software*, (Pisa, Italy), pp. 165–174, ACM Press, 2004.

[22] E. M. Goldratt, *What is this thing called Theory of Constraints and how should it be implemented?* Great Barrington, MA: North River Press, 1990.

[23] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. New York: ACM Press, 1998.

[24] D. Stewart, R. Volpe, and P. Khosla, "Design of dynamically reconfigurable real-time software using port-based objects," *Software Engineering, IEEE Transactions on*, vol. 23, no. 12, pp. 759–776, 1997.

[25] C. Becker, G. Schiele, H. Gubbels, and K. Rothermel, "Base - a micro-broker-based middleware for pervasive computing," in *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*, pp. 443–451, 2003.

[26] P. Bellavista, A. Corradi, and C. Stefanelli, "Corba solutions for interoperability in mobile agent environments," in *Distributed Objects and Applications, 2000. Proceedings. DOA '00. International Symposium on*, pp. 283–292, 2000.

[27] A. R. Tripathi, T. Ahmed, and N. M. Karnik, "Experiences and future challenges in mobile agent programming," *Microprocessors and Microsystems*, vol. 25, no. 2, pp. 121–129, 2001.

[28] M. Li, H. Wang, and P. Li, "Merging sn and masp to build up pervasive computing infrastructure," in *TENCON '02. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, vol. 1, pp. 400–403 vol.1, 2002.

[29] G. Vigna, "Mobile agents: ten reasons for failure," in *Mobile Data Management, 2004. Proceedings. 2004 IEEE International Conference on*, pp. 298–299, 2004.

[30] L. Avramopoulos and M. Anagnostou, "Optimal component configuration and component routing," *Mobile Computing, IEEE Transactions on*, vol. 1, no. 4, pp. 303–312, 2002.

[31] J. Williams and N. Bergmann, "Programmable parallel coprocessor architectures for reconfigurable system-on-chip," in *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pp. 193–200, 2004.